Minimal information for reusable scientific software

Minimal information for reusable scientific software

Neil Chue Hong
Software Sustainability Institute
JCMB, Mayfield Road
Edinburgh, EH9 3JZ, UK
N.ChueHong@software.ac.uk

ABSTRACT

One of the biggest challenges for developers of scientific software is understanding how best to make the software *reusable*. A particular problem is that the concept of reusability combines many different concerns, including *whether* the software can be reused, *how* it can be reused, and by *whom*. This paper looks at the concept of software reusability from the perspective of the software engineer and the researcher. It proposes a multi-level framework for improving the reusability of scientific software, which minimises the information and effort required such that it is easier for scientific software developers, who are often researchers, to provide appropriate levels of information to support reuse.

Keywords

Scientific software, research software, software reuse, software reusability, software sustainability, minimal information standards.

1. INTRODUCTION

Software underpins much of the scientific research undertaken today. As well as the "traditional" use of software for modelling and simulation, it is used to manage and control instruments, and analyse and visualise data. An incredible amount of investment of effort and money is put into scientific software. In the UK, the Engineering and Physical Sciences Research Council (EPSRC) estimated that it had invested approximately £9m per annum on scientific software [1]. A major concern for funders is the amount of reuse of software developed under their investment: given the cost of software production there is an onus on it being usable by the widest possible set of users. Yet each year, researchers and developers will choose to create a piece of software from scratch rather than reuse an existing one. How can we address this issue?

1.1 Challenges and perceptions

One of the biggest challenges for developers of scientific software is understanding how best to make the software *reusable*. A particular problem is that the concept of reusability combines many different concerns, including *whether* the software can be reused, *how* it can be reused, and by *whom*.

Whether a piece of software can be reused depends on three things: the "quality" of the software, the "ability" of the re-user, and any restrictions placed on the software. In software engineering terms [2], the quality of the software encompasses both *quality of design* (the degree to which the software meets the functional requirements) and *quality of conformance* (the degree to which the software as implemented meets the non-functional requirements such as robustness and maintainability). The ability of the re-user is goes hand in hand with the quality of the software Restrictions on the software include things like the license the software has been released under, but also whether the software can be discovered and understood to meet the re-users' requirements.

Additionally, there are many motivations that disbenefit reuse. In the research sector, there is a focus on novelty and originality, such that maintenance, improvement and even quality are not seen as priorities for a developer. As a researcher, you may not want to use someone else's code because you do not believe it does everything you want, or even because you would prefer the fun of developing it yourself. What is more, the diversity and variety of software used in a research environment often mean that personal recommendations and demonstrations at conferences are often the only way to get a sense of the reusability of the software. Therefore a major challenge is the provision of adequate levels of information to support discovery and understanding of the capabilities of the software, without placing an undue burden on the developer.

Ultimately, software reusability is important not just for correctness: it enables improved efficiency and productivity, ability to link related outputs, and a more sustainable research software ecosystem. Researchers should be able to understand which software they should choose to reuse or modify for their work. What is required is a simple way of describing a piece of software to let others reuse it.

2. SOFTWARE REUSE

2.1 The software engineering perspective

The topic of software reuse has been considered from a software engineering perspective for many decades. From the first cost model for software reuse [3] developed at CMU's Software Engineering Institute by Holibaugh et al in 1989 , there have been successive iterations of research (Koltan and Hudson's Reuse Maturity Model, the reuse model for DARPAs STARS program [4], the CMMI [5], the SSMM [6], QSOS[1]) into improving the reusability of software code and developing *maturity models* to describe the code.

Reusability can be applied at many layers: software, platforms, libraries, components, APIs, code, formats, models. Most models concentrate on defining a process for assessing the reusability of the code itself. One of the best known examples is the NASA Reuse Readiness Levels [7][8]. This allows others to easily assess the reuse potential (from limited reusability to having demonstrated extensive reusability) across a number of topics (including documentation, packaging, licensing and portability). This gives a comprehensive and comparative framework for assessing the reusability of scientific codes. However this is not a small undertaking, and it is still aimed at the software engineer rather than all researchers that develop software.

[1] http://www.qsos.org/

2.2 The researchers perspective

The difference between scientific software development and general software development can be illustrated by one fundamental characteristic: in general, software is written by a developer for others to use, however with scientific software it is often (initially) written by a researcher to use themselves. Further, whilst there are various incentives for developers in general to make their software reusable (e.g. commercial returns, reputation) there are currently few for researchers who develop software. There are exceptions to this of course: larger projects which hire research software engineers because they recognise the need to share software with collaborators; "cottage industry" developers seeking to capitalise by producing software based around a new research technology or technique that they can sell to researchers.

In all cases, the "economics" of reusability are different: because the incentives are fewer and because the developer of research software is often primarily developing it for use in their own research, the cost benefit analysis would suggest that the software should be made only as good and as reusable as necessary to get to the next publication. Nevertheless, given the increased emphasis on open science and reproducible research, many researchers would like something that provides an answer to C. Titus Brown's idea of the Ladder of Academic Software Reusability + Sustainability[2], or this author's own Five Stars of Research Software[3].

In recent years, various suggestions and discussion around the topic of the information required for reusability, each with an emphasis on the unique issues of scientific software, have taken place. Notably, these include the Significant Properties of Software study[4] [9], the Science Code Manifesto[5], Code as a Research Object[6], and the 1st WSSSPE workshop[7] [10][11]. The NSF has encouraged new initiatives in this area[8] through a Dear Colleague letter on Supporting Scientific Discovery through Norms and Practices for Software and Data Citation and Attribution. The Journal of Open Research Software[9] publishes software metapapers which contain basic information about pieces of scientific software. Each of these seek to reach a pragmatic balance between reducing the effort of the developer to make the software reusable and the effort of the user to reuse it. However, thus far there has not been an attempt to create a multi-level framework that gives the developer a choice of information to provide, along with the corresponding benefit for reuse.

[2] http://ivory.idyll.org/blog/ladder-of-academic-software-notsuck.html

[3] http://www.software.ac.uk/blog/2013-04-09-five-stars-research-software

[4] http://www.stfc.ac.uk/e-Science/projects/medium-term/software-preservation/22428.aspx

[5] http://sciencecodemanifesto.org/

[6] https://github.com/mozillascience/code-research-object/issues/2

[7] http://wssspe.researchcomputing.org.uk/wssspe1/

[8] http://www.nsf.gov/pubs/2014/nsf14059/nsf14059.jsp

[9] http://openresearchsoftware.metajnl.com/

3. A FRAMEWORK FOR INFORMING THE REUSE OF SCIENTIFIC SOFTWARE

To promote the reuse of scientific software, whilst balancing the amount effort available to most developers of scientific software, I have defined a framework for informing the reuse of scientific software. This framework is split into four levels (Levels 1-4) which I consider to be what all developers of scientific software should be aiming to meet. Each level builds on the one below, to gradually build up the information that is required to promote reusability. Therefore a developer can start at level 1 and progress to more advanced levels if additional time and effort are available, if they perceive the need to promote reuse, or there is evidence of issues coming from others attempting to reuse the software. I also note two additional levels (Level 0 and Level 5) which I consider not to be useful as they represent, respectively, the theoretical minimum and idealistic minimum which either provide insufficient information for the user or place too high a barrier on the developer.

The different pieces of information are split into five categories:

- LICENSE: the legal constraints on reusability on the software
- AVAILABILITY: information relating to the discovery and accessibility of the software
- QUALITY: information relating to understanding the functional and non-functional requirements fulfilled by the software
- SUPPORT: information relating to how the user may communicate with the developers
- INCENTIVE: information that enables developers and users to be rewarded for reuse

3.1 Level 0: Theoretical Minimum

This is the theoretical minimum requirement for reusability, but would not be considered to promote reusability.

- LICENSE: the software has a license.
- AVAILABILITY: the software is available via some mechanism.

3.2 Level 1: Absolute Minimum

This is a practical absolute minimum requirement that places no barrier on the developer. It should also be considered the absolute minimum information to be provided by *any* researcher who has published a paper that includes results produced by software they have written:

- LICENSE: the software has a license that allows reuse (this can include non-Open Source licences that allow reuse under academic or commercial terms).
- AVAILABILITY: the software has been published somewhere such that people can find it (this could be as a tarball on a website).
- QUALITY: the software has some minimal indication of what it is supposed to do (e.g. "This software finds and sorts variants in a file containing genetic modifiers"), normally as part of a README.
- SUPPORT: the software indicates some way of contacting the original/current developer (in lieu of good documentation), normally as part of a README.

3.3 Level 2: Useful Minimum

This is a useful minimum level of information which does not place significant additional effort over what might be expected to support their own use of the software.

- All of the information from Level 1, plus:
- AVAILABILITY: the software is in a repository of some form, including the source code in a code repository if this is made available.
- QUALITY: the software has enough documentation to understand how to run it without contacting the original developer. This would normally include sample input and output files, and basic options/parameters.
- QUALITY: the documentation says what combination of dependencies (software and hardware) the developer believes the software requires.
- INCENTIVE: some way of citing/attributing the developers is provided.

3.4 Level 3: Pragmatic Minimum

This is a pragmatic level of information which I consider most developers should strive to provide. It requires some additional effort over the previous levels, but not significantly more than would be required if simply collaborating on the code development with another developer.

- All of the information from Level 2, plus:
- LICENSE: the software has a licence that allows modification as well as reuse.
- AVAILABILITY: source code for the software should be in a code repository under version control and commit messages should be minimally useful.
- AVAILABILITY: the software is published via a website which includes details of what the software does and is indexed by search engines.
- QUALITY: the software describes some form of running tests in an automated fashion.
- QUALITY: the software provides at least one automated "system test" including input, output and parameter data which enable a user to run the software through a complete pipeline/workflow example.
- QUALITY: each major package / subroutine should have some documentation. Code documentation should be about design and scientific purpose, not the mechanics of the code.
- SUPPORT: the software has an associated mailing list, issue tracker, or similar mechanism for raising and resolving issues.
- INCENTIVE: there is a DOI attached to the software / a paper about the software that enables it to be cited using traditional mechanisms.

3.5 Level 4: Good Minimum

This is a good minimum level of information which actively encourages reuse of software but requires a slightly higher level of curation of the information.

- All of the information from Level 3, plus:
- LICENSE: the software has an OSI approved open source licence that allows modification as well as reuse.
- LICENSE: any data accompanying the software is released under a Creative Commons license that allows reuse.
- AVAILABILITY: the reuse information defined in this framework is presented in a machine readable form.
- QUALITY: the software uses an automated test framework and has reasonable test coverage.
- QUALITY: the software lists dependencies including languages, versions, operating systems and formats. Links to dependencies are provided if not bundled with software.
- QUALITY: the documentation describes the type of people that would be expected to use the software, and provides step-by-step examples and screenshots of how they would use it.
- SUPPORT: the documentation includes basic information on how to extend and modify the software.
- SUPPORT: the software can be built and/or installed using simple, automated procedures.
- SUPPORT: the software documentation includes the developers own perceived definition of reusability.
- INCENTIVE: all contributors to the software are acknowledged.

3.6 Level 5: Idealistic "Minimum"

In discussions, various other suggestions have been made which have been advocated as "minimum" requirements for reuse. In each case, they represent the ideal implementation of some software engineering practice. However I believe these not only place such a demand on the developer that they will chose not to make any concessions for reusability but are also are not required by most users reusing the software.

Examples include:

- 100% unit test coverage
- Automatic binary installers for multiple platforms
- Use of a dependency manager

It should also be noted that these are not strictly *information*, but rather requirements on the project. It is important that we do not expect every project to reach the idealistic minimum or none will reach the useful minimum.

4. CONCLUSIONS

Scientific software is increasingly important for all areas of research, and significant investment is made to support its development. Yet often software is not reused by others who could benefit which means the value of the investment is diminished, and the total investment required increases. Whilst many proposals have been made to define a set of information that should be provided to enable reusability, these are often aimed at a single level or fail to take into account the required balance between the effort required of the developer to make the software reusable and the effort required of the user to understand how to reuse it.

By providing a multi-level framework for defining the information that needs to be provided to promote reuse of scientific software, I hope that this paper provides a pragmatic way of encouraging developers of all levels to improve the reusability of their software.

5. ACKNOWLEDGMENTS

My thanks to Kevin Ashley, Martin Fenner, Ross Gardler, David Shotton, Kenji Takeda, and people at the EPSRC Software Strategy Town Meeting who inspired the original idea behind the Five Stars of Research Software; to Caitlin Bentley, Adam Crymble, Barry Rowlingson, Robin Wilson, and Mark Woodbridge who contributed refinements to the Five Stars; to Carl Boetigger, C. Titus Brown, Robert Davidson, Konrad Hinsen, Karthik Ram, Kaitlin Thaney, Greg Wilson and the other contributors to the Code as a Research Object project for helping to shape my thinking around the framework; to Arfon Smith for his ideas around README files; and finally my colleagues at the Software Sustainability Institute for providing the inspiration and examples that informed the development of the framework, in particular Steve Crouch, Mike Jackson, Simon Hettrick and Aleksandra Pawlik. The Software Sustainability Institute is supported by EPSRC grant EP/H043160/1.

6. REFERENCES

[1] EPSRC. 2012. *Software as an Infrastructure.* Accessed on 19th July 2014 from: http://www.epsrc.ac.uk/newsevents/pubs/software-as-an-infrastructure/

[2] Pressman, S. 2010. *Software Engineering: A Practitioners Approach (7th Edition).* p398-406.

[3] Holibaugh, R et al. 1989. *Reuse: where to begin and why.* Proceedings of the conference on Tri-Ada '89: Ada technology in context: application, development, and deployment. p266-277. DOI: 10.1145/74261.74280.

[4] Frazier, T.P., and Bailey, J.W. 1996. *The Costs and Benefits of Domain-Oriented Software Reuse: Evidence from the STARS Demonstration Projects.* Accessed on 21st July 2014 from: http://www.dtic.mil/dtic/tr/fulltext/u2/a312063.pdf

[5] CMMI Product Team, 2006. *CMMI for Development, Version 1.2.* SEI Identifier: CMU/SEI-2006-TR-008.

[6] Gardler, R. 2013. *Software Sustainability Maturity Model.* Accessed on 21st July 2014 from: http://oss-watch.ac.uk/resources/ssmm

[7] NASA Earth Science Data Systems Software Reuse Working Group (2010). *Reuse Readiness Levels (RRLs), Version 1.0.* April 30, 2010. Accessed from: http://www.esdswg.org/softwarereuse/Resources/rrls/

[8] Marshall, J.J., and Downs, R.R. 2008. *Reuse Readiness Levels as a Measure of Software Reusability.* In proceedings of Geoscience and Remote Sensing Symposium. Volume 3. P1414-1417. DOI: 10.1109/IGARSS.2008.4779626.

[9] Matthews, B. et al. 2010. *A Framework for Software Preservation.* International Journal of Digital Curation. DOI: 10.2218/ijdc.v5i1.145

[10] Stodden, V and Miguez, S 2014. *Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research.* Journal of Open Research Software 2(1):e21, DOI: 10.5334/jors.ay

[11] Venters, C.C., Lau, L, Griffiths, M.K., Holmes, V, Ward, R.R., Jay, C, Dibsdale, C.E. and Xu, J 2014. *The Blind Men and the Elephant: Towards an Empirical Evaluation Framework for Software Sustainability.* Journal of Open Research Software 2(1):e8, DOI: 10.5334/jors.ao

[12] Chue Hong, N., Hole, B. and Moore, S. 2013. *Software Papers: improving the reusability and sustainability of scientific software.* Contribution to 1st Workshop on Sustainable Scientific Software: Practice and Experience. DOI: 10.6084/m9.figshare.795303